UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 2

Entity Classes

Date Due: May 22, 2019, 8pm **Total Marks: 112**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- **Make sure your name and student number appear at the top of every document you hand in.** These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- **Assignments must be submitted to Moodle.**
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- **Programming questions must be written in Java.**
- **Non-programming questions must be submitted as either `.txt` or `.pdf` files.** If other file types are used, you will receive a grade of zero for that question.

Version History

- **07/15/2019**: released to students

Question 1 (112 points):

Purpose: Practising Object-Oriented Programming

Degree of Difficulty: Tricky

For this question, you must implement three classes: a `Person` class, a `BasicDoctor` class, and a `Ward` class. Each class should have a constructor, as well as the instance variables and methods specified below. For each of the classes, all the instance variables should be declared as private. Be sure to follow the specification given below, since the classes will be used in subsequent assignments in order to build a simple hospital management system. In addition, each class is to have a static main method that should be used to test the class methods following the principles of testing as discussed in class.

Note from your instructor: This assignment is big. I would highly recommend you employ the test-driven development strategy you used in CMPT 145. It will save you a lot of time and unnecessary effort debugging later on.

(a) (24 points) The `Person` class. This class models a person. For our purposes, a person will have the following features:

- A name
- A health card number
- A constructor that takes in the person's name and health card number
- An accessor method for the name
- A mutator method for the name
- An accessor method for the health card number
- A toString() method that returns a string representation of all the information about the person in a form suitable for printing
- A main method that will test all of the above features

(b) (18 points) The `BasicDoctor` class. This class is the representation for a doctor. While it is anticipated that a doctor will have a number of attributes that could be stored, we will keep this basic version very simple. It will have the following features:

- A name
- A constructor that takes in the doctor's name
- An accessor method for the name
- A mutator method for the name
- A toString() method that returns a string representation of all the information about the doctor in a form suitable for printing
- A main method that will test all of the above features

(c) (46 points) The `Ward` class is represents a physical ward in a Hospital (like the emergency room). It will have the following features:

- The ward name
- The integer label of the first bed
- An array that stores the people in the beds of the ward (this will be modified to a Container Class in a later assignment)
- A constructor that takes in parameters for the ward name, the first integer label for a bed in the ward, and the last integer label for a bed in the ward; **See the section below titled "Additional Information" for more insight into how the bed labels should work**

- An accessor method for the ward name
- An accessor method for the first bed label
- A method that returns the last bed label
- A method that converts a bed label to its corresponding array index
- A method that converts an array index to a bed label
- A method that checks to see if a particular bed label is occupied
- A method that gets the person at a particular bed label, the bed must be non-empty
- A method that assigns a person to a particular bed label, the bed must be empty
- A toString() method that returns a string representation of all the information about the ward in a form suitable for printing including the name of the ward and for each bed:
  - the bed label (if the bed is empty) OR
  - the bed label and the name of the person in the bed
- A main method that will test all of the above features

hint `double[] example = new double[10];` will create an array of doubles with size 10.

(d) (24 points) You will need to include internal and external documentation with every class. See below for what is expected:

- Proper internal documentation includes:
  - A comment just before the class header that gives an overview of the class. For example, if the class models an entity, the comment might state what entity the class models and specify the key features. Alternatively, if the class serves as a container, state what type of items the container stores, and how these items are accessed. If the class is an interface, state what it provides an interface for and whether there is anything special about the interface. Finally, if it has a control function, what is it doing and controlling? Recall that comments for a class appear before the class and begin with `/**`.
  - A comment just before each instance variable stating what is stored in the field, again beginning with `/**`
  - A comment before each constructor and method stating what it does. Note that the comment only gives what the routine does, not how it does it. If it isn?t obvious from the code how it accomplishes its goal, comments on how it is done belong in the body of the method.
  - Single line comments as needed to improve the readability of your code. Over-use of single-line comments will result in a deduction of marks.
  - Use descriptive variable names and method names.
  - Include good use of white space, especially reasonable indentation to improve the readability of your code.
  - Hard-code all of the data for your testing. Minimize console input and output by only reporting errors(These are the same principles we applied in CMPT 145).

- External Documentation:
  - A description of how to execute your tests of the classes. What is necessary to compile your classes? What program or programs need to be run to show the execution of your program? For example, it might be necessary to compile and execute all three classes individually. On the other hand, you might have written a driver program (another class with a static main method that invokes the main methods of the three required classes; the static main method of class `A` can be invoked by `A.main(null);`). In the latter situation, we need to know the name of the driver class. This description should be very short.

- The status of your assignment. What is working and what is not working? What is tested and what is not tested? If it is only partially working, the previous point should have described how to run that part or parts that work. For the part or parts not working, describe how close they are to working. For example, some of the alternatives for how close to working are (i) nothing done; (ii) designed but no code; (iii) designed and part of the code; (iv) designed and all the code but anticipate many faults; or (v) designed and all the code but with a few faults; (vi) working perfectly and thoroughly tested.
- Maintenance manual. This is information for the person or persons who must keep your system running, fix any faults, and do any upgrades. The reader of the maintenance manual is expected to have the same background and experience as yourself, but of course not having developed your system. This part of the documentation will vary from assignment to assignment. For this assignment, it is sufficient to include a UML class diagram showing all the features of each class, and the relationships (inheritance, uses and aggregation) amongst the classes (if any).
- UML diagrams should be generated with a program like UMLet or an online tool, such as https://www.gliffy.com/uses/uml-software/. Make sure that you use the correct arrows and arrowheads. Incorrect arrowheads will lose several marks. You can also draw out a UML diagram by hand and scan it. Just make sure it is easily readable.

Additional information

Bed Labels As further information about the integer labels of the beds, suppose that the first bed has label 200 and the last one has label 250. Recall that in Java, the indices of an array always start at 0. Thus, the array indices will be 0, 1, 2, ..., 50. In addition, integer bed label 200 will need to map to array index 0, 201 map to 1, etc. All classes outside the Ward class will specify a bed by its integer label (say 200 ... 250), while the Ward class must also deal with array indices. All the complications in dealing with array indices should be within the Ward class. Hence, if later a more complex scheme is used to label beds, all the changes to convert the new labels to array indices will be in the Ward class.

Assignment Scope The above classes are simply entity classes for use in a larger system. A hospital application would have a whole other part (perhaps quite large) to obtain information and requests from users, to determine what operations are to be done, to carry out those operations by invoking methods of the entity classes, to return appropriate information back to the user, etc. Later assignments will fill in these parts. You do not need them for this assignment.

What to Hand In

- The completed `Person.java` program.
- The completed `BasicDoctor.java` program.
- The completed `Ward.java` program.
- `a2q1_documentation.pdf` that contains the external documentation for each class and the application as a whole. Any documentation relating to a specific class should be easily identifiable.

Be sure to include your name, NSID, student number and course number at the top of all documents.

Evaluation

In each class, the following evaluation scheme is used:

2 pts For each instance variable

2 pts For each method

2 pts For testing each method

3 pts For internal documentation

5 pts For external documnetation